CS 415: Programming Languages

Homework 3: Prolog

Due Friday, 7 October by 10 a.m.

In this assignment, you will write a Prolog program that can perform symbolic differentiation with reduction of the answer to a simple form. There are 2 main parts to this assignment: symbolic differentiation and expression simplification.

Part 1: Symbolic Differentiation

In mathematics, symbolic differentiation is an operation that converts a given arithmetic expression into another arithmetic expression called the derivative. Suppose *U* stands for an arithmetic expression which may contain a variable *x*. The derivative of *U* with respect to *x* is written as d*U*/d*x*, and is defined recursively by applying some conversion rules (shown below) to expression *U*.

Two boundary conditions appear first. The arrow in the rules is read 'is converted to'. *U* and *V* stand for expressions, and *c* for a constant:

d*c*/d*x* -> 0	
d*x*/d*x* -> 1	
d(-*U*)/d*x* -> -(d*U*/d*x*)	
d(*U*+*V*)/d*x* -> d*U*/d*x* + d*V*/d*x*	
d(*U*-*V*)/d*x* -> d*U*/d*x* - d*V*/d*x*	
d(*cU*)/d*x* -> c(d*U*/d*x*)	
d(*UV*)/d*x* -> *U*(d*V*/d*x*) + *V*(d*U*/d*x*)	
d(*U*/*V*)/d*x* -> d(UV^{-1})/d*x*	in other words: d(*U*/*V*) is the same as d(*U* * (*V*^(-1)))
d(U^c)/d*x* -> cU^{c-1}(d*U*/d*x*)	written as `U^c`, `U^(c-1)`, etc.
d($\log_e U$)/d*x* -> U^{-1}(d*U*/d*x*)	written as `log(x)`

Consider each conversion rule as a goal of the form d(*E*,*X*,*F*) which succeeds when the derivation of expression *E* with respect to variable *X* is the expression *F*. For example, the following questions may be asked of d after it is defined:

```
?- d(x+1,x,X).
X = 1+0

?- d(x*x-2,x,Y).
Y = 1*x+1*x-0
```

Part 2: Expression Simplification

Notice that simply transforming one expression into another using the rules does not necessarily produce the result in a simplified form. You should write an algebraic simplifier as a separate procedure so that the final answer is in a simple form according to the rules specified below.

1. Addition/Subtraction
    a. Addition/Subtraction of 0 to any variable should reduce to the variable itself (or its negative in the case of subtraction)
        - E.g. *x*+0 should print as *x* and 0-*x* as -*x*
    b. Addition/Subtraction of 2 integers should reduce to their sum/difference
        - E.g. 4+7 should print as 11 and 4-7 as -3
2. Multiplication
    a. Multiplication of 0 with any constant or variable should reduce to 0
        - E.g. 0**x*, 7*4*0, or *x*^2*0 should all print 0
    b. Multiplication of 1 with any constant or variable should reduce to that constant or variable
        - E.g. 1*2**x*^3 should print 2**x*^3
    c. Multiplication of 2 integers should reduce to their product
        - E.g. 3*4 should print 12
    d. Multiplication of 2 integers with a variable should be reduced to its simplest form
        - E.g. 2*4**x* and 2**x*4 (more on this later) should print 8**x*
3. Exponentiation
    a. Exponentiation of any variable to 1 should reduce to that variable itself
        - E.g. *x*^1 should print *x*
    b. Exponentiation of any two numbers should reduce to their corresponding result
        - E.g. 2^3 should print 8

For example, your program should print the following results for the inputs specified:

```
?- d(2*x^3+3*x^2-7*x+4,x,Y).
Y = 6*x^2+6*x-7

?- d(2*x^2-log(3*x),x,Y).
Y = 4*x-3* (3*x)^ -1
```

What to turn in

Please include a comment at the top of your file that includes your name and email ID. Also please include comments in your code to aid the TA in understanding any tricky parts in your code. Please submit your code in a single file.